(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 546 863 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number : **92311359.1**

(51) Int. Cl.^{5} : **H03M 7/30**

(22) Date of filing : **11.12.92**

(30) Priority : **13.12.91 US 807007**

(43) Date of publication of application :
16.06.93 Bulletin 93/24

(84) Designated Contracting States :
CH DE FR GB IT LI NL SE

(71) Applicant : **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504 (US)

(72) Inventor : **Craft, David John**
114 Bluff Park Circle
Austin, Texas 78746-4300 (US)

(74) Representative : **Atchley, Martin John Waldegrave**
IBM United Kingdom Limited Intellectual Property Department Hursley Park
Winchester Hampshire SO21 2JN (GB)

(54) **Data compression apparatus.**

(57) The present invention relates to data compression apparatus comprising means (320) for receiving an input string of data words, storage means (330) for storing selected data words, comparison means for comparing the input data words with the stored data words, token generating means (370) for generating tokens representing input data words which are identical to stored data words, and means for combining the tokens with input data words which are not identical with any stored data words to form a compressed string of data words.

According to the invention the data compression apparatus is characterised in that the storage means comprises means for storing input data words which are not identical to any stored data words so as to increase the number of stored data words.

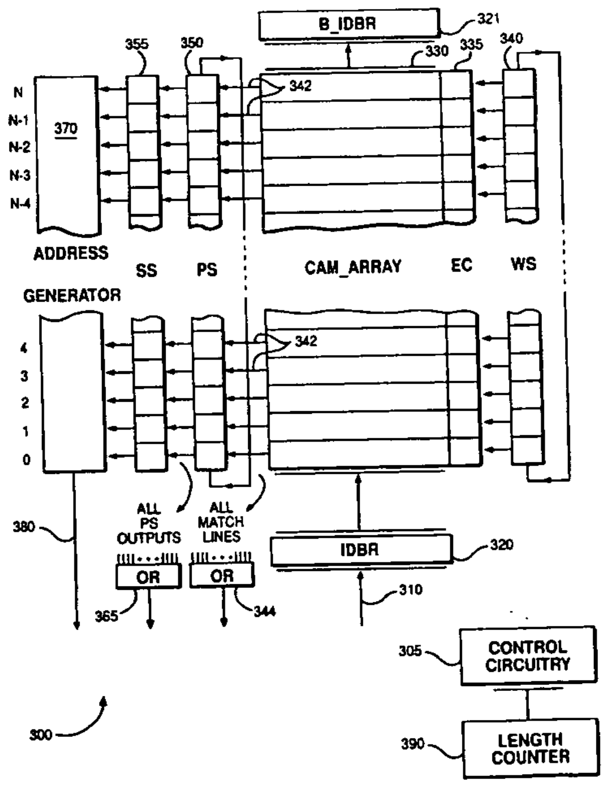


FIG. 4

EP 0 546 863 A2

The present invention is directed to an apparatus for compressing and decompressing strings of data words.

Many types of data compression systems exist. One commonly used technique is the Lempel-Ziv algorithm which is described in "Compression of Individual Sequences via variable Rate Coding" by Lempel and div in IEEE Transactions on Information Theory, Sept., 1977, pages 530-536. Figs. 1A-1C of the accompanying drawings illustrate a typical implementation of the Lempel-Ziv algorithm. In Fig. 1A, a shift register 10 that is N+1 bytes long is used to temporarily store previously processed data bytes or words. If a new data string to be processed includes at least one data byte that has been processed before, then a token including the length and relative address of the previously processed data byte in the shift register will be generated. This can in general be expressed using fewer bits of information than the data byte itself, so the data string is effectively compressed. If a data byte to he processed does not form part of a previous data byte existing in the shift register, then a token or tokens will be generated containing this data byte explicitly. In general, such tokens have to be expressed using slightly more bits of information than the data byte itself, so there is an effective expansion. Overall, the gain from the compressed data bytes usually exceeds the losses from the non-compressed data bytes, so overall data compression results. If there are no repeating bytes of data in the data string, then the data string cannot be compressed by this technique.

Fig. 1B illustrates the generation of a token referencing previously processed data. In the example given, the data bytes A, B, C and D were previously processed and are currently stored in the shift register at addresses 37, 36, 35 and 34. Input bytes to be processed are A, B, C and E. The new data includes the string of bytes ABC that has a length of 3 and matches previously stored string of bytes ABC at relative address 37. The address is relative because once a token is generated describing the string, the bytes A, B, and C will be loaded into the shift register and the bytes A, B, C and D will be shifted down the shift register to a new address. The address of a data byte in the shift register is relative to the number of data bytes subsequently processed.

Fig. 1C illustrates the generation of a second token referencing a previously stored data byte. In the example given, the values A, B, C and Z are to be processed. The input data includes the string of bytes ABC that has a length of 3 and matches previously stored string of bytes ABC at relative addresses 3 and 41. The token generated in this example is usually the lower relative address of 3. Tokens include the count and relative address of the previously processed string of bytes and are expressed as (count, relative address). As a result of the compression of the bytes A, B, C, E, A, B, C and Z as shown in Figs. 1B and 1C, the generated processed output data will include: (3, 37), E, (3, 3), Z.

One of the primary problems with implementations of the Lempel-Ziv compression technique is the difficulty in performing the search operation for previous matching strings of bytes at an effective processing speed. Many techniques discussed below are modifications of the Lempel-Ziv technique that attempt to improve the speed of the technique by improving the speed of the search operation or the amount of compression achieved by using more efficient token encoding.

US -A- 4,021,782 teaches a compaction device to be used on both ends of a transmission line to compact, transmit, and decompact data. Each possible incoming character is categorised according to an expected frequency use in a preset coding table. The category of the character affects the encoding of that character (more frequently used characters have shorter generated code).

US -A- 4,464,650 discloses parsing an input data stream into segments by utilising a search tree having nodes and branches.

US -A- 4,558,302 teaches what is commonly called a Lempel-Ziv-Welch data compression technique. This patent discloses utilising a dictionary for storing commonly used data strings and searching that dictionary using hashing techniques.

US -A- 4,612,532 discloses interchanging positions of candidates or ordering the table of candidates in approximate order of frequency.

US -A- 4,622,585 describes compression of image data. This patent discloses how a first series of data bits in a current picture line and a second series of data bits of a preceding picture line are shifted together in a compression translator.

US -A- 4,814,746 describes utilising a dictionary for storing commonly used data strings and deleting from that dictionary data strings not commonly used.

US -A- 4,853,696 describes a plurality of logic circuits elements or nodes connected together in reverse binary treelike fashion to form a plurality of logic paths corresponding to separate characters.

US -A- 4,876,541 is directed to improvements to the Lempel-Ziv-Welch data compression technique described above by using a novel matching algorithm.

US -A- 4,891,784 is directed to an auto blocking feature in a tape environment which utilises a packet assembly/disassembly means for auto blocking.

US -A- 4,899,147 is directed to a data compression technique with a throttle control to prevent data underruns and an optimising startup control. This patent discloses a capability to decompress data read in a reverse direction from which is was written. This patent also discloses a string table for storing frequently used strings with a counter of the number of times the string has been used.

US -A- 4,906,991 is directed to copy codeword encoding utilising a tree data structure.

US -A- 4,988,998 is directed to preprocessing a string of repeated characters by replacing a sequentially repeated character with a single character and repeat count.

The object of the present invention is to provide improved data compression apparatus for compressing a string of data words.

The present invention relates to data compression apparatus comprising means for receiving an input string of data words, storage means for storing selected data words, comparison means for comparing the input data words with the stored data words, token generating means for generating tokens representing input data words which are identical to stored data words, and means for combining the tokens with input data words which are not identical with any stored data words to form a compressed string of data words.

According to the invention the data compression apparatus is characterised in that the storage means comprises means for storing input data words which are not identical to any stored data words so as to increase the number of stored data words.

In order that the invention may be more readily understood an embodiment will now be described with reference to the accompanying drawings in which:-

Figs. 1A-1C are diagrams of a prior art technique for compressing data,

Figs. 2 and 3 are illustrations of various system configurations utilising a preferred embodiment of the invention,

Fig. 4 is a block diagram illustrating a preferred data compression engine,

Fig. 5 is a block diagram illustrating a single section of the compression engine,

Figs. 6A-6C is a flowchart describing a preferred process utilised to run the compression engine, and

Fig. 7 is a block diagram illustrating a preferred decompression engine.

A major difficulty with a hardware decoder implementation of the Lempel-Ziv technique discussed above is the use of a relative addressing scheme. Such a scheme requires the use of a shift register to hold previously processed data bytes or words, one word in each data element. Each incoming data word is shifted into the first position of the shift register while all the previously processed data words are shifted into adjacent positions. In addition, a random access capability is required to each element of the shift register. This requires much more circuitry, chip area, and power to implement than a simple random access memory.

The embodiment to be described uses a fixed addressing scheme that allows the stored previously processed data to remain in a fixed location in memory. This permits a very simple, fast hardware decoder implementation using a simple random access memory and two address counters. For the encoder, a correspondingly simple organisation is proposed, using a context addressible memory, or CAM, for storage of the previously processed data rather than a random access memory or RAM. This significantly decreases the search overhead requirements for each word operation while performing an exhaustive string matching process. This improves the compression ratio, as well as allowing very fast encoder throughput. An additional benefit is that a large part of both the encoder and decoder are commonly used structures, lending themselves well to VLSI implementation in silicon.

Figs. 2 and 3 are illustrations of various system configurations for the embodiment being described. As shown in Fig. 2, a computer 100 includes a central processing unit (CPU) 105 that communicates with system memory 110. The CPU also communicates on bus 112 with input/output channels or adapters 115 and 120. Through the input/output channels, the CPU may communicate with other computer systems 125, tape drives 130, disk drives 135, or other input/output devices 138 such as optical disks. The computer 100 may also include a compression/decompression engine 140 on bus 112. The compression/decompression engine (unit) includes compression engine 141 and decompression engine 142. These engines may be invoked by an operating system file handler running on the CPU to do compression or decompression of data being transmitted or received through the input/output channels. The engines may utilise system memory 110 or an optional memory 145 while performing the desired compression or decompression of data. If optional memory 145 is used, the compressed or decompressed data may be transmitted directly to and from the I/O channels on optional bus 147.

Fig. 3 illustrates a computer 150 including a CPU 155 and system memory 160. The CPU communicates on bus 162 with input/output channels or adapters 165 and 170. Through the input/output channels, the CPU may communicate with other computer systems 175, tape drives 180, disk drives 185 or other input/output devices 188. Coupled to the input/output channels are compression/decompression engines (units) 190 and 194 for compressing and decompressing some or all data passing through the input/output channels. The compression/decompression engines include compression engines 191, 195 and decompression engines 192, 196.

The engines may also have optional memories 198 and 199 for working as buffers and for handling space management tasks as the data is compressed or decompressed.

There are many other alternative system configurations which would be apparent to those of ordinary skill in the art. For example, one computer system, such as a server, may include a data compression engine for compressing all data sent to it while the remaining computer systems may each include a decompression engine to decompress all data they receive from the server.

Fig. 4 is a block diagram illustrating a preferred embodiment of data compression engine 300. The operation of the various elements of the data compression engine are controlled by a control circuit 305 which is coupled to each of the elements described below. The control circuit is a logic circuit which operates as a state machine as will be described in greater detail below. Data enters the compression engine on input data bus 310 into input data buffer register (IDBR) 320 and is later stored in a backup input data buffer 321. The data stored in the input data buffer register is then compared with all current entries in a content addressable memory (CAM) array 330.

The CAM array includes many sections (N+1 sections are shown in the figure), each section including a register and a comparator (as will be described in greater detail in Fig. 5 below). Each CAM array register stores a word of data and includes a single storage element 335, called an empty cell EC, for indicating whether a valid or current data word is stored in the CAM array register. Each comparator generates a match or active signal when the data word stored in the corresponding CAM array register matches the data word stored in the input data buffer register 320 (plus some other criteria described below).

Coupled to the CAM array is a write select shift register (WS) 340 with one write select cell for each section of the CAM array. At any selected time, there is a single write select cell set to 1 or active while the remaining cells are set to 0 or inactive. The active write select cell selects which section of the CAM array will be used to store the data word currently held in the input data buffer register. The write select shift register will then be shifted one cell to determine which CAM array section will store the next data word in the input data buffer register. The use of the write select shift register provides for many of the capabilities of the above described prior art Lempel-Ziv shift register while also allowing the use of fixed addresses.

Current entries in the CAM array are designated as those CAM array sections that have empty cells (EC) set to 1 and have corresponding write select (WS) cells set to 0. If the data in any of the current CAM entries matches the data word in the input data buffer register, then the comparator for that CAM array section will generate an active match signal on its corresponding one of the match lines 342. As a result, a match OR gate 344, which is coupled to each of the match lines, will generate a match signal.

A primary selector shift register (PS) 350 is used to indicate those corresponding sections in the CAM array where the matching operation has been successful so far. That is, a PS cell which is set to 1 (active) indicates which CAM array section contains the last word of a matching string, or sequence, of data words. More than one cell in the PS column may be active because there could be a number of sections in the CAM array where the incoming string of data words has occurred before. If the incoming stream of data words has not occurred before, then raw (uncompressed) data tokens are generated so as to contain the stream of data words until a stream of data words is found that does match previously processed data.

Prior to loading another new data word into the input data buffer register 320, the preceding entry in the input data buffer register is saved in backup input data buffer register (B_IDBR) 321. In addition, the contents of the column of PS cells is saved in a corresponding secondary selector (SS) register 355. Furthermore, a length counter 390 is incremented by 1 to maintain a length of the matching string. Finally, the primary selector PS and write selector WS cells are shifted one location so that the entry for the Mth PS cell is now stored in the (M+1)th PS cell and the entry for the Mth WS cell is now stored in the (M+1)th WS cell. This shift operation accomplishes two things. First, it shifts the single active write selector entry one cell so the next incoming data word is written into the next sequential section of the CAM array. Secondly, it shifts the active cells in the primary selector column so they each correspond to the CAM array sections immediately subsequent to the previous matching CAM array sections in the sequence of the array sections.

Once a new incoming data word has been loaded into the input data buffer register 320, it is compared with all the current entries in the CAM array as described above. The corresponding match lines will be active for all CAM array data cell entries that match the new data word. If any of the match lines is active, then the match OR gate 344 will be active. All active PS cells whose corresponding match lines are not active are cleared. If any of the PS cell entries remains active, indicating that the current data word in the input data buffer register extends the previous matching string of data words, then a PS OR gate 365 coupled to all of the PS cells will be active. Therefore, the next data word can be processed to determine whether it may also continue the matching string of data words stored in the CAM array. This matching process continues until there is a 0 at the output of the PS OR gate, signifying that there are now no matches left. When this occurs, the values marking the end points of all the matching strings existing prior to this last data word are still stored in the SS cells. Address generator 370 then determines the location of one of the matching strings and generates its address on bus 380. The address generator 370 may be a logic circuit utilised to generate an address given a signal from one or more of the cells of the secondary selector register 355. Such a logic circuit may be easily designed by one of ordinary skill in the art. The length of the matching string has been tracked by length counter 390.

The address generator 370 generates the fixed (also known as absolute) address of the section of the CAM array 330 containing the end of the matching string and the length counter 390 provides the length of the matching string. Therefore, a start address and length of the matching string can be calculated, encoded and generated as a token. This process is described in greater detail below.

One advantage of using a fixed address to store data words is that the data words do not need to be shifted from register to register, which could require additional circuitry. The arrangement being described uses a shifting wraparound write select cell to determine which fixed address CAM array section to write information to and utilises the features of a content addressable memory to do the many needed comparisons to find matches with previous strings of data. As a result, the arrangement being described performs compression more quickly and efficiently.

The arrangement being described also provides for more effective compression than many existing Lempel-Ziv techniques because the search for matching data strings is exhaustive. That is, the CAM array 330 allows for an exhaustive search for all possible matching strings. Many existing techniques use a compromising technique such as hashing in order to reduce the search time but which may not find the longest matching string in memory.

Fig. 5 is a block diagram illustrating a single section of the compression engine 300. This section 500 includes a write select cell 510, a CAM array section 525, a primary selector cell 550 and a secondary selector cell 560. The CAM array section 525 includes a comparator 540 and a register 530 which includes an empty cell 520. Each of these elements is coupled to the control circuitry 305 described above.

A reset signal line 600 is utilised by the control circuitry to clear the write select cell 510 and the empty cell 520 to 0 when the reset signal line is active. A shift signal line 610 is utilised to shift the value in the write select cell 510 and the primary selector cell 550 to the next CAM array section as well as to receive the value stored in the previous CAM array section. If the Mth cell is the last cell in the CAM array, then the value in the cell will be wrapped around to the first cell in the CAM array. WS IN and WS OUT lines 611 and 612 and PS in and PS OUT lines 613 and 614 carry the values in the WS and PS cells to and from the subsequent and previous WS and PS cells. A load signal line 620 is utilised to load the value from the primary selector cell 550 to the secondary selector cell 560. A write signal line 630 is utilised to initiate writing of data to the CAM array register 530 as well as to turn the empty cell 50 to a value of 1. An input data buffer line 640 is utilised to transfer data from the input data buffer register IDBR to the CAM array register 530 during a writing operation or to the comparator whenever a comparison is performed.

If the write selector cell is already active (set to a value of 1), a write signal to the write selector cell 510 will cause the write select cell to generate a signal to the empty cell and the CAM array register. The empty cell will then be set to 1 and the CAM array register will have the data on the input data buffer register bus 640 written into the CAM array register. The value on the input data buffer register bus and the hard coded 1 and 0 are compared by comparator 540 to the value in CAM array register 530, the value of empty cell 520 and the value of write select cell 510. If the write select cell is set to 0 and the empty cell is set to 1 and the value on the input data buffer register bus is equal to the value in the CAM array register, then comparator 540 will generate a matched signal to the primary selector. Otherwise the comparator will generate a non-matched signal to the primary selector.

In a preferred embodiment, there are 2,048 sections in the CAM array, thereby requiring the use of at least 11 bits to designate the address of a string within the CAM array. The CAM array may be 2,048 sections connected in parallel or it may be 4 sets of 512 sections coupled in parallel. In an alternative embodiment, the CAM array is 1,024 sections deep, thereby requiring at least a 10 bit address description or 512 bits requiring at least a 9 bit address. Alternative embodiments allow the use of different size CAM arrays.

Figs. 6A-6C is a flowchart describing the process utilised to run the compression engine 300. In a preferred embodiment, the flowcharted process is accomplished with 21 states. The flowchart will now be described with reference to the 21 states.

In a first state 00 and step 800, the compression engine is initialised by clearing the length counter 390 to 0, clearing all the primary selector cells 350, clearing all the empty cells 335 in the CAM array to 0, clearing all the write select cells 340 to 0 and setting the last write select cell (N) to 1. Because the empty cells are all cleared to 0, the entries in the CAM array registers do not need to be cleared, thereby saving time and decreasing power requirements. Execution then proceeds to state 01.

In state 01 and step 810, if an input word is not available on the input data bus 310 for loading into the input data buffer register 320, then execution will proceed to state 15, otherwise execution will proceed to state 02. This is to handle the "end of file" condition wherein there are no more input data words to be compressed.

In state 02 and steps 810-860, the data word in the input data buffer register 320 is moved to the backup input data buffer register 321 and the next data word on the input data bus is read into the input data buffer register. In addition, the data in the primary selector 350 is loaded into the secondary selector 355 to store the results of any previous string matching operations that have occurred so far. Furthermore, the primary selector cells PS and write selector cells WS are shifted one position. The write select cells are shifted one position to designate the next CAM array cell to be written to (only a single write select cell has a value of 1). The primary selector cells are shifted one position in preparation for the next compare operation. This is to check whether the next input data word will match any of the data words in the CAM array that follow previously successful string matches. Finally, a search/write function is performed on the CAM array. This function first compares the data word in the input data buffer register 320 with each of the data words stored in each of the sections in the CAM array that have a corresponding empty cell set to 1 (designating that the CAM array cell contains data) and a write select cell set to 0 (designating that the cell is not the next one to be written to). If there are any matches, then the corresponding match lines are set to 1, and the corresponding primary selector cells remain at 1 if they were previously set to 1. In the case of the first input data word, there are no previously matched strings as indicated by all the primary selector cells set to 0. If any match lines or primary selector cells are set to 1, then the primary selector OR gate will generate a value of 1. Subsequently, the data word in the input data buffer register 320 is written to the CAM array register 330 with a corresponding write select cell equal to 1. Execution then proceeds to state 03.

In state 03 and step 870, execution proceeds to state 08 if the output of the primary selector OR gate 365 is 0, otherwise execution proceeds to state 04. This is to determine whether a previously matched string continues to match entries in the CAM array.

In state 04 and step 880, execution proceeds to state 07 if the length counter is less than its maximum value; otherwise execution proceeds to state 05. In the preferred embodiment, the maximum value is 271 due to the coding used to generate the tokens that describe the string as will be described in greater detail below. In alternative embodiments, the maximum value may be based on the maximum value the length counter can store.

In state 05 and step 890, a compressed word token is generated. The token has a length value equal to the length counter value minus 2. The token also has a displacement value equal to the address of the lowest set secondary selector cell (indicating the cell with the last matching word to the input data string) minus the length counter value plus 1 (indicating the address at the first matching word in the CAM array to the input data string). Execution then proceeds to state 06.

In state 06 and steps 900-910, the primary selector cells 350 are set to the values of the corresponding match lines 342. This clears the previous PS values and starts the next possible matching string. In addition, the length counter 390 is cleared to 0. Execution then proceeds to state 07.

In state 07 and step 920, the length counter 390 value is incremented by one and execution is returned to state 01 for starting work on the next data word.

State 08 is executed if the current data word in the input data buffer register 320 does not continue to match any string of data words in the CAM array 330. In state 08 and step 930, execution proceeds to state 10 if the length counter 390 value is less than two. If the length counter value is two or more, then the data word in the input data buffer register ends a previous string of matching data words.

In state 09 and step 940, a compressed word token is generated. As described in state 05, the compressed word token has a length value equal to the length counter 390 value minus 2 and a displacement value or length equal to the address of the lowest set secondary selector cell 350 minus the length counter value plus 1. Execution then proceeds to state 10. In the alternative, execution could proceed directly to state 14.

In state 10 and step 950, execution proceeds to state 12 if the length counter value is not equal to one, otherwise execution proceeds to state 11.

In state 11 and step 960, a raw word token (indicating a word has not been compressed) is generated using the data value in the backup data buffer register 321. Execution then proceeds to state 12. In the alternative, execution could proceed directly to state 14.

In state 12 and step 970, execution proceeds to state 14 if the output from match OR gate is equal to one; otherwise execution proceeds to state 13.

In state 13 and steps 980-990, a raw word token is generated using the data value on the backup data buffer register 321. The length counter 390 is then cleared to 0. Execution then proceeds to state 01.

In state 14 and steps 994-996, the primary selector cells 350 are set equal to the values of the corresponding match lines 342. This is to start the matching of a new string of words. In addition, the length counter 390 is cleared to 0. Execution then proceeds to state 01.

States 15-20 are executed if there are no more data words on the input data buffer register (see state 01). In state 15 and step 1000, if the length counter value is less than 2, then execution proceeds to state 18, otherwise execution proceeds to state 16.

In state 16 and step 1010, the values in the primary selector cells 350 are loaded into the secondary selector cells 355. Execution then proceeds to state 17.

In state 17 and step 1020, a compressed word token is generated as previously shown in states 05 and 09. The compressed word token has a length value equal to the length counter value minus 2 and a displacement value equal to the address of the lowest set secondary selector cell minus the length counter value plus 1. Execution then proceeds to state 18. In the alternative, execution could proceed directly to state 20.

In state 18 and step 1030, if the length counter value is equal to 0, then execution proceeds to state 20; otherwise execution proceeds to state 19.

In state 19 and step 1040, a raw word token is generated using the data value in the backup data buffer register 321. Execution then proceeds to state 20.

In state 20 and step 1050, an end marker token is generated.

Fig. 7 is an illustration of a preferred decompression engine 1500 to decompress data generated by the above described compression engine 300. Control circuitry 1510 receives the compressed data stream including raw word tokens (a data word preceded by a 0), compressed word tokens (a length of a string and a start address) and an end marker token (e.g. 13 1s in a row).

If a raw word token is received, the enclosed data word is stored in register 1520. The contents of the register are then written to a history buffer 1530 (with the same number of sections as the CAM array used to compress the data) at the address in a write counter 1540 (initially set to 0) through address multiplexer 1550. In addition, the content of the register 1520 is also generated as an output data word. The write counter 1540 is then incremented by one. If the write counter value is already at N, then it is set to 0. As a result of this operation, each data word is written to the next subsequent section in the history buffer 1530, thereby mirroring the CAM array used during data compression, and the data word is also generated as output.

If a compressed word token is received, the address is loaded into a read counter 1560 and the length is loaded into token length counter 1570. The data in the history buffer array addressed by the read counter through multiplexer 1550 is then read into register 1520. That data word is then written back into the history buffer at the address in the write counter 1540 to continue the mirroring of the CAM array used to compress the data. The data word in the register 1520 is also generated as an output data word. The write counter 1540 is then incremented by one as described above. If the token length counter 1570 value is greater than one, then it is decremented by one, the read counter 1560 is incremented by one, and the process repeats for the next data word. This process repeats until the whole data string referred to as the compressed data string has been sequentially read from the history buffer, written back into the history buffer, and generated as output.

An optional subtract unit 1580 may be included to allow this decompression engine to decompress Lempel-Ziv compressed data that utilises relational addresses as described in Figs. 1A-1C. This subtract circuit would be used to convert the relational address to a fixed address by subtracting the value in the write counter 1540 from the relational address. The subtract circuit could also be included but disabled when decompressing data compressed by the arrangement described above and enabled when decompressing Lempel-Ziv compressed data using relational addresses.

In the preferred embodiment, a raw word token is generated as a 0 followed by the raw (uncompressed) word. A compressed word token is passed as a 1 followed by the length of the matching string and the starting location of the matching string in the CAM array (called the displacement). A control token may also be generated which starts with eight 1's and is followed with four bits designating the control instructions. Finally, an end token is passed to designate the end of a compressed data stream. The end token is thirteen 1's in a row.

Table 1 shows the codes used in the preferred embodiment to designate the length of a compressed data word string. This type of coding is a modified logarithmic coding wherein shorter strings utilise shorter codes and longer strings utilise longer codes. This is a useful coding technique when the frequency of shorter strings is substantially greater than the frequency of longer strings. The displacement is specified with an 11 bit value in the preferred embodiment. A shorter displacement may be used with a CAM array having fewer sections.

TABLE 1: Codes Used to Designate Compressed Word

Code Field	Compressed Word Length
00	2 words
01	3 words
10 00	4 words
10 01	5 words
10 10	6 words
10 11	7 words
110 000	8 words
... ...	... ...
... ...	... ...
110 111	15 words
1110 0000	16 words
....	... ...
....	... ...
1110 1111	31 words
1111 0000 0000	32 words
....	... ...
....	... ...
1111 1110 1111	271 words

In order to provide for future expansion of the arrangement described above, control instructions may be passed in the compressed data stream. These control instructions may include instructions such as "reset the history buffer", "preload the history buffer with a preferred data set", etc. In a preferred arrangement, there are two types of control instructions, long and short. Table 2 illustrates long control instructions wherein a 12 bit control field is given followed by an 11 bit control subfield. This provides for 2048 subfields for each of the four control fields for a total of 8208 possible long instructions.

TABLE 2: Long Instruction Control Fields and Subfields

Long Control Field	Control Subfield
1111 1111 0000	0000 0000 000 - 1111 1111 111
1111 1111 0001	0000 0000 000 - 1111 1111 111
1111 1111 0010	0000 0000 000 - 1111 1111 111
1111 1111 0011	0000 0000 000 - 1111 1111 111

Table 3 illustrates the short control instructions. The short control instructions are only 12 bits long and are, therefore, fewer in number than the total number of long control instructions. However, the short control instructions require less time to transmit. As described above, one short control field has already been defined as an end marker. The end marker is a 1 (defining the following bits as being either a compressed data token or as a control instruction) followed by the twelve bit end marker control instruction (twelve 1s).

TABLE 3 : Short Instruction Control Fields

Control Field	Current Function
1111 1111 0100	not defined
1111 1111 0101	not defined
1111 1111 0110	not defined
....	... .
....	... .
1111 1111 1110	not defined
1111 1111 1111	end marker

Claims

1. Data compression apparatus comprising
   means (320) for receiving an input string of data words,
   storage means (330) for storing selected data words,
   comparison means (540) for comparing said input data words with said stored data words,
   token generating means (370) for generating tokens representing input data words which are identical to stored data words, and
   means for combining said tokens with input data words which are not identical with any stored data words to form a compressed string of data words,
   characterised in that said storage means comprises means for storing input data words which are not identical to any stored data words so as to increase the number of stored data words.

2. Data compression apparatus as claimed in Claim 1 characterised in that said storage means comprises a plurality of storage cells each capable of storing a selected data word, and means for storing each selected data word in a respective one of said storage cells,
   in that said comparison means comprises means for comparing each input data word with the data word stored in each of said cells in succession, and
   in that said storage means also comprises means for storing any input data word that is not identical to any stored data word in a selected empty storage cell.

3. Data compression apparatus as claimed in Claim 3 characterised in that said storage cells are arranged in a sequence and said selected storage cell is the next empty cell in the sequence.

4. Data compression apparatus as claimed in either Claim 2 or Claim 3 characterised in that said token generating means comprises means for generating a token for an input data word which is based on the address of the cell in said storage means in which is stored the data word identical to said input data word.

FIG. 1A
PRIOR ART

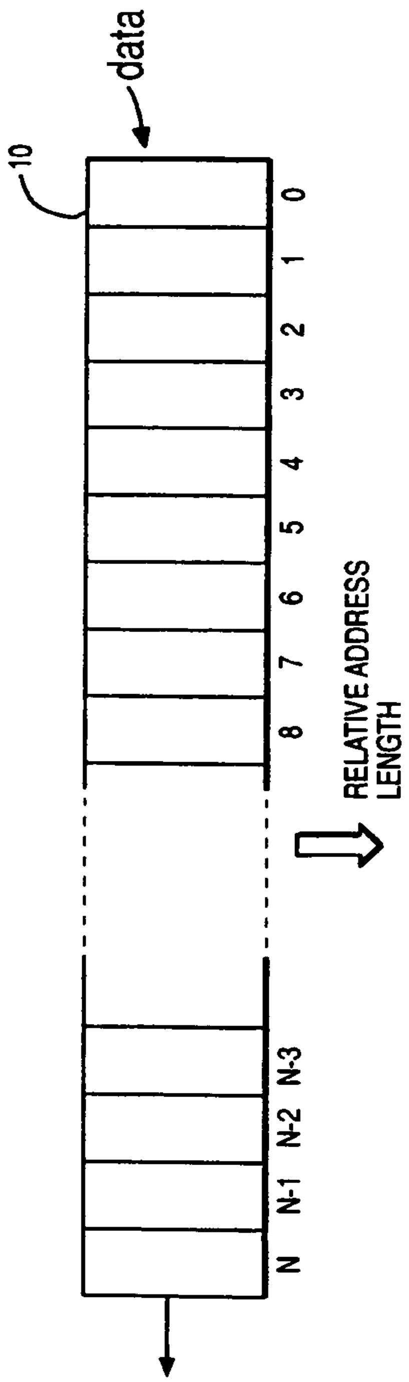


FIG. 1B
PRIOR ART

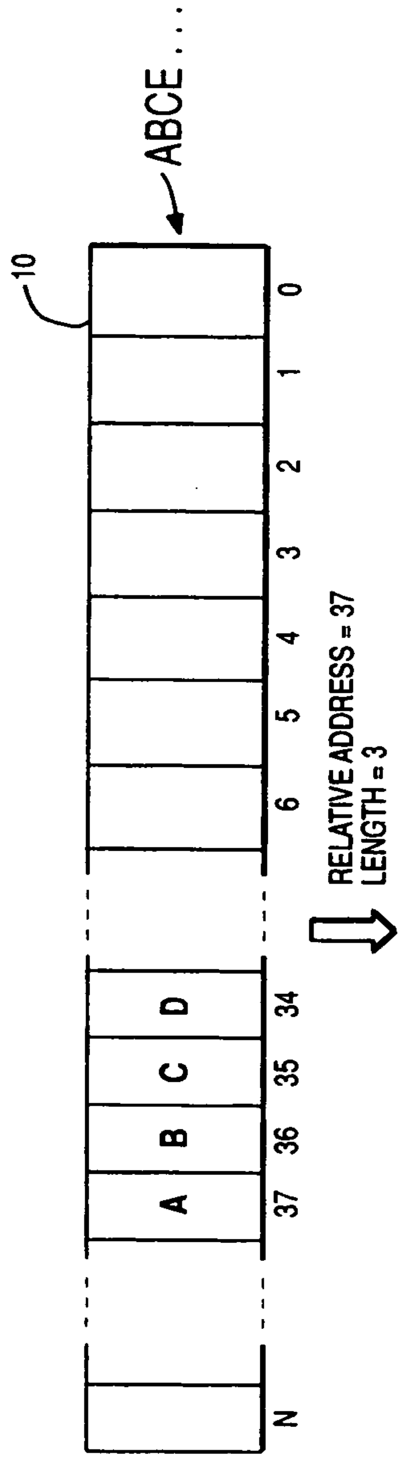
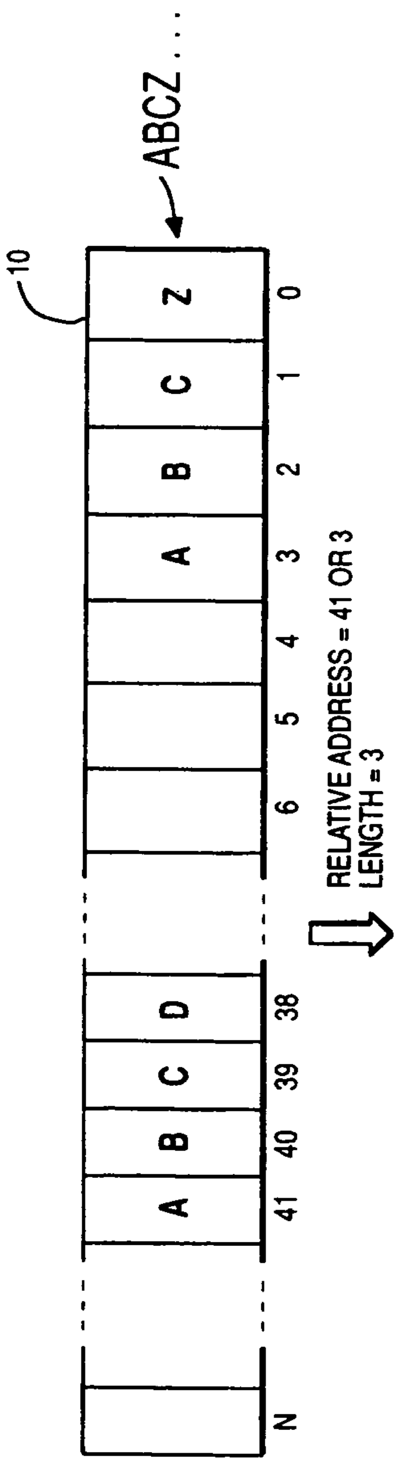


FIG. 1C
PRIOR ART



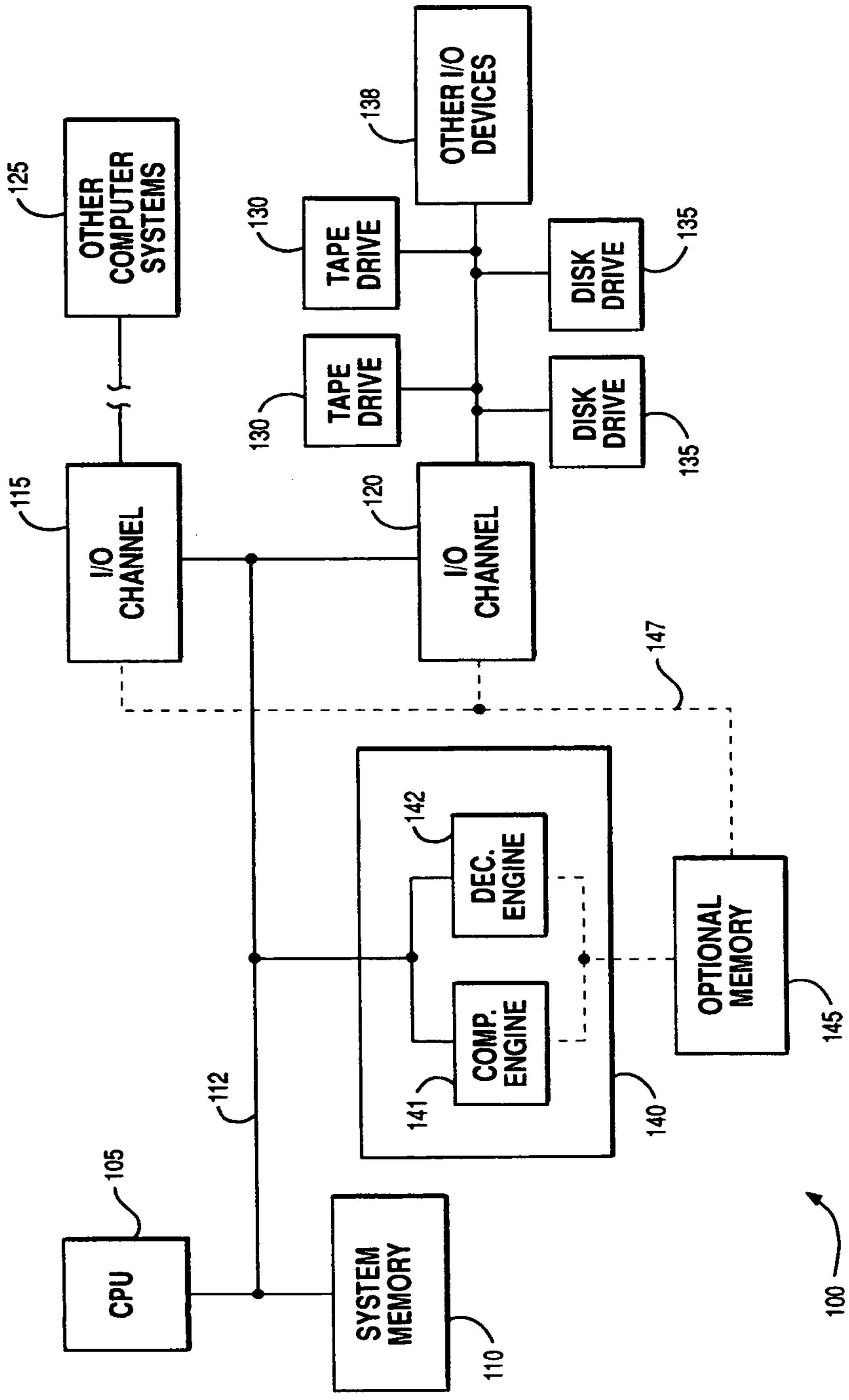
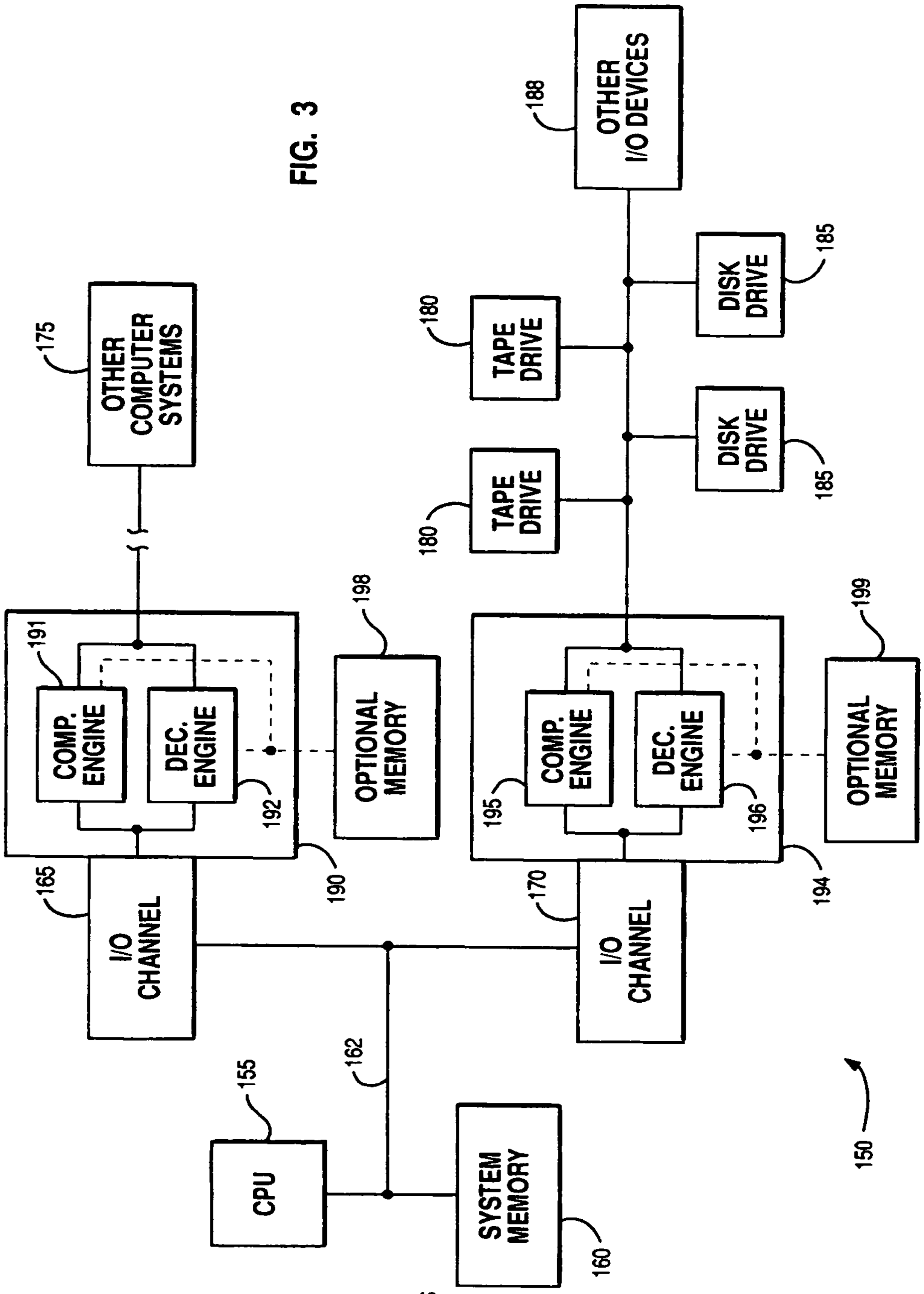

OTHER COMPUTER SYSTEMS
125
OTHER I/O DEVICES
138
TAPE DRIVE
130
TAPE DRIVE
130
DISK DRIVE
135
DISK DRIVE
135
I/O CHANNEL
115
I/O CHANNEL
120
147
DEC. ENGINE
142
COMP. ENGINE
141
140
OPTIONAL MEMORY
145
112
CPU
105
SYSTEM MEMORY
110
100

FIG. 2

FIG. 3


CPU
155
SYSTEM MEMORY
160
162
I/O CHANNEL
165
I/O CHANNEL
170
COMP. ENGINE
191
DEC. ENGINE
192
190
OPTIONAL MEMORY
198
OTHER COMPUTER SYSTEMS
175
COMP. ENGINE
195
DEC. ENGINE
196
194
OPTIONAL MEMORY
199
TAPE DRIVE
180
TAPE DRIVE
180
DISK DRIVE
185
DISK DRIVE
185
OTHER I/O DEVICES
188
150

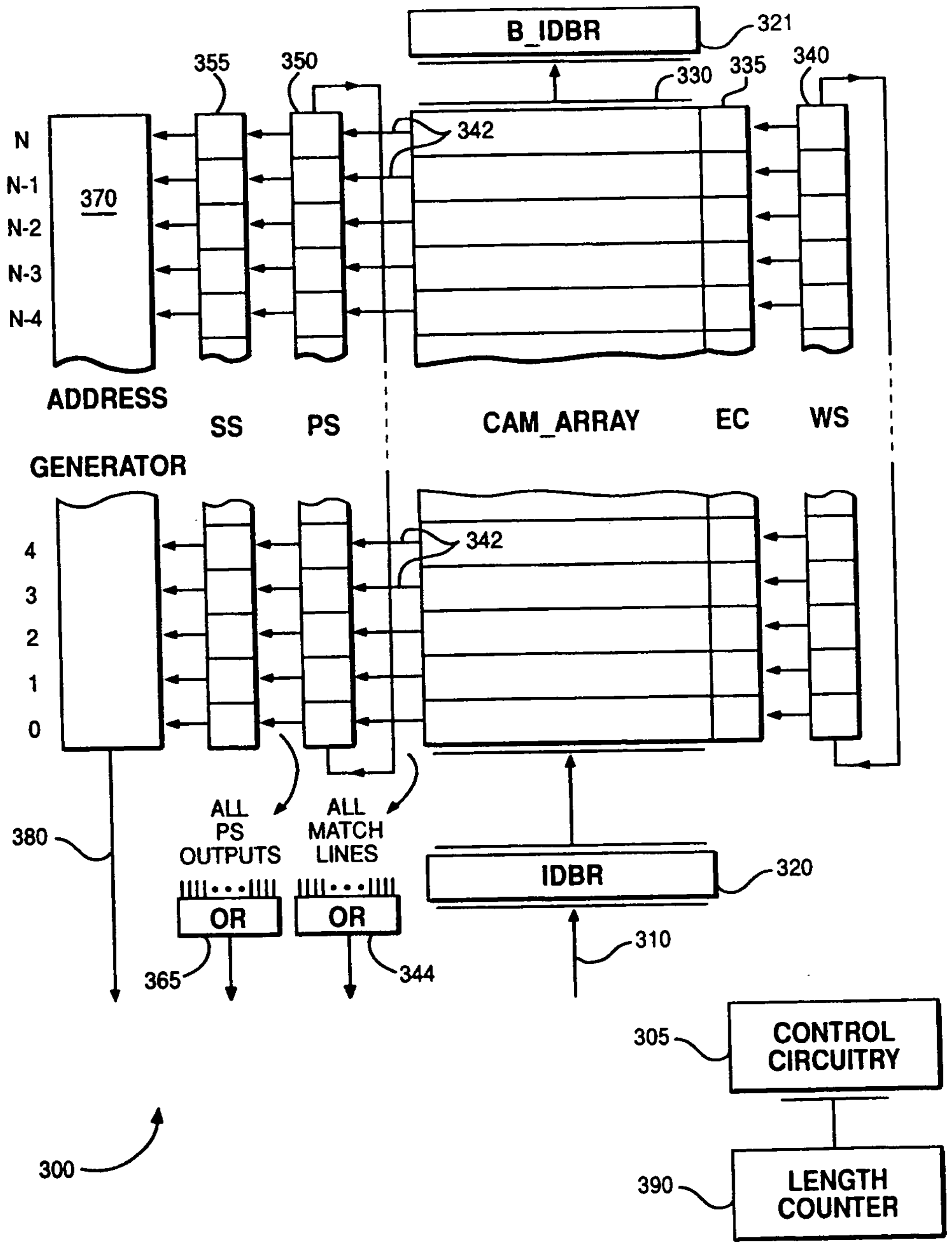

B_IDBR
321
355
350
330
335
340
342
N
N-1
N-2
N-3
N-4
370
ADDRESS
GENERATOR
SS
PS
CAM_ARRAY
EC
WS
342
4
3
2
1
0
ALL PS OUTPUTS
ALL MATCH LINES
OR
OR
IDBR
320
310
380
365
344
305
CONTROL CIRCUITRY
390
LENGTH COUNTER
300

FIG. 4

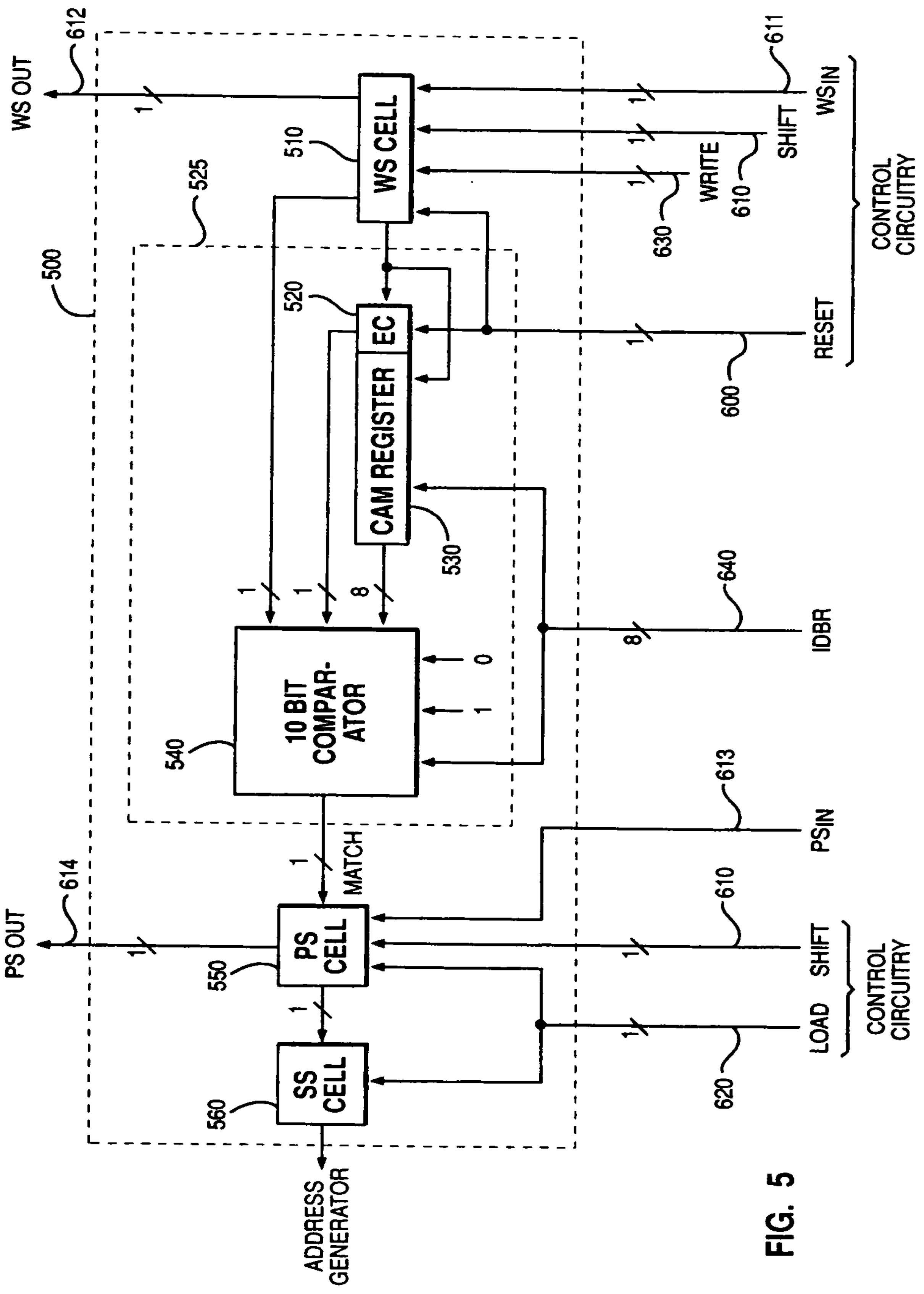

WS OUT
612
1
WS CELL
510
525
500
520
EC
CAM REGISTER
530
1
1
8
10 BIT COMPAR-ATOR
540
0
1
MATCH
1
PS OUT
614
1
PS CELL
550
1
SS CELL
560
ADDRESS GENERATOR
611
WSIN
1
SHIFT
610
1
WRITE
630
1
CONTROL CIRCUITRY
RESET
600
1
640
IDBR
8
613
PSIN
610
SHIFT
1
LOAD
620
1
CONTROL CIRCUITRY

FIG. 5

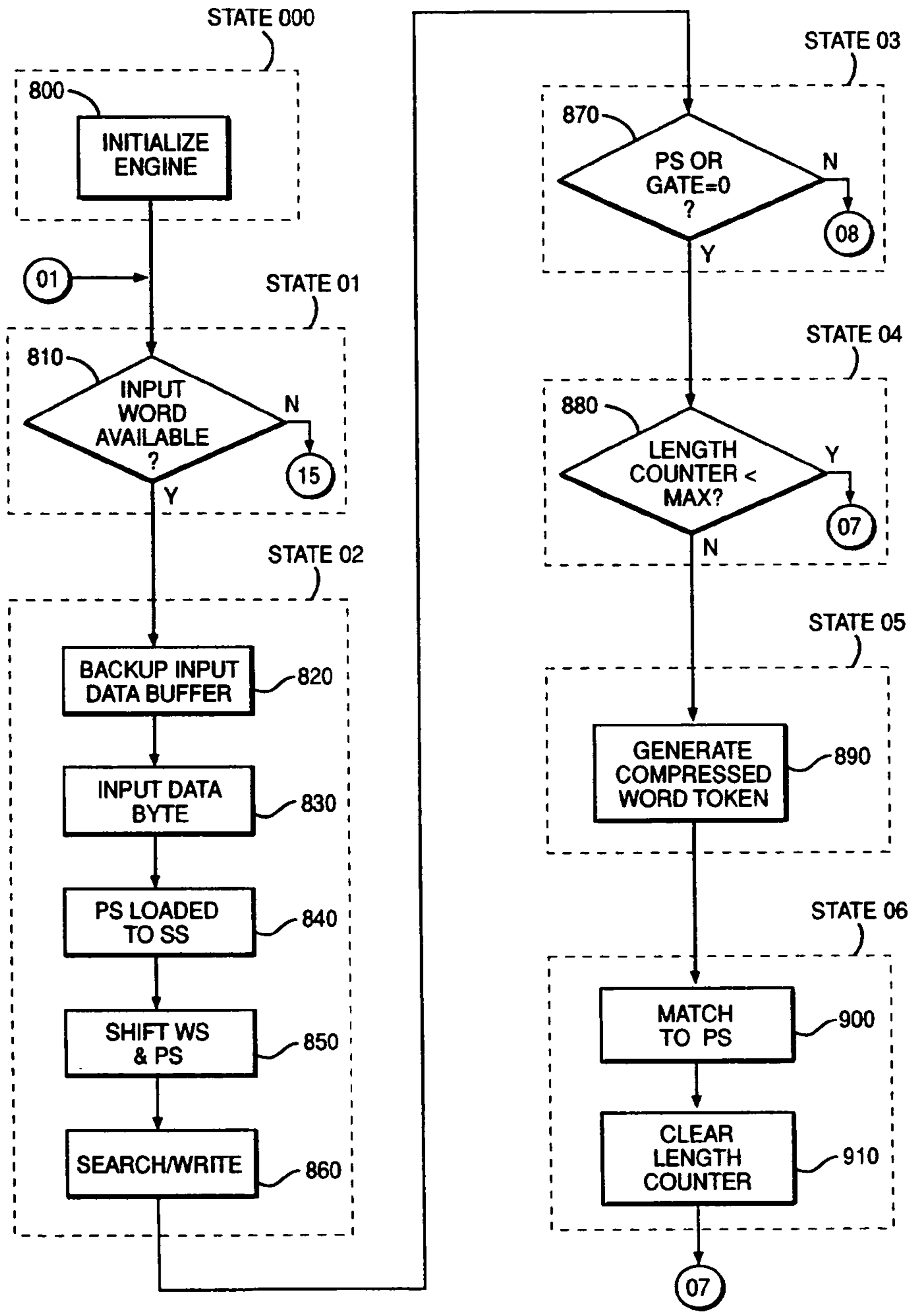

STATE 000
800
INITIALIZE ENGINE
01
STATE 01
810
INPUT WORD AVAILABLE ?
N
15
Y
STATE 02
BACKUP INPUT DATA BUFFER
820
INPUT DATA BYTE
830
PS LOADED TO SS
840
SHIFT WS & PS
850
SEARCH/WRITE
860
STATE 03
870
PS OR GATE=0 ?
N
08
Y
STATE 04
880
LENGTH COUNTER < MAX?
Y
07
N
STATE 05
GENERATE COMPRESSED WORD TOKEN
890
STATE 06
MATCH TO PS
900
CLEAR LENGTH COUNTER
910
07

FIG. 6A

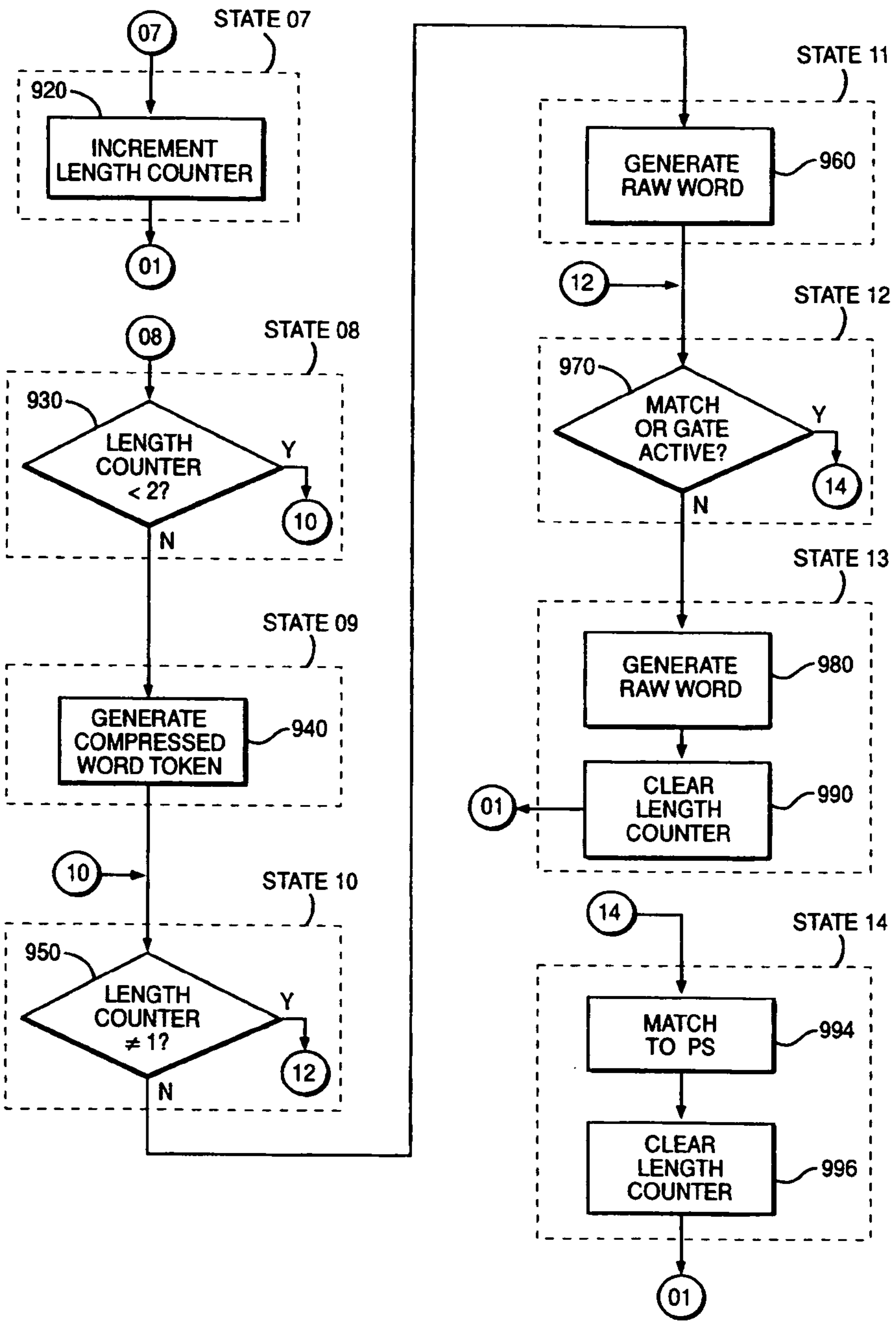

STATE 07
07
920
INCREMENT LENGTH COUNTER
01
STATE 08
08
930
LENGTH COUNTER < 2?
Y
10
N
STATE 09
GENERATE COMPRESSED WORD TOKEN
940
10
STATE 10
950
LENGTH COUNTER ≠ 1?
Y
12
N
STATE 11
GENERATE RAW WORD
960
12
STATE 12
970
MATCH OR GATE ACTIVE?
Y
14
N
STATE 13
GENERATE RAW WORD
980
CLEAR LENGTH COUNTER
990
01
14
STATE 14
MATCH TO PS
994
CLEAR LENGTH COUNTER
996
01

FIG. 6B

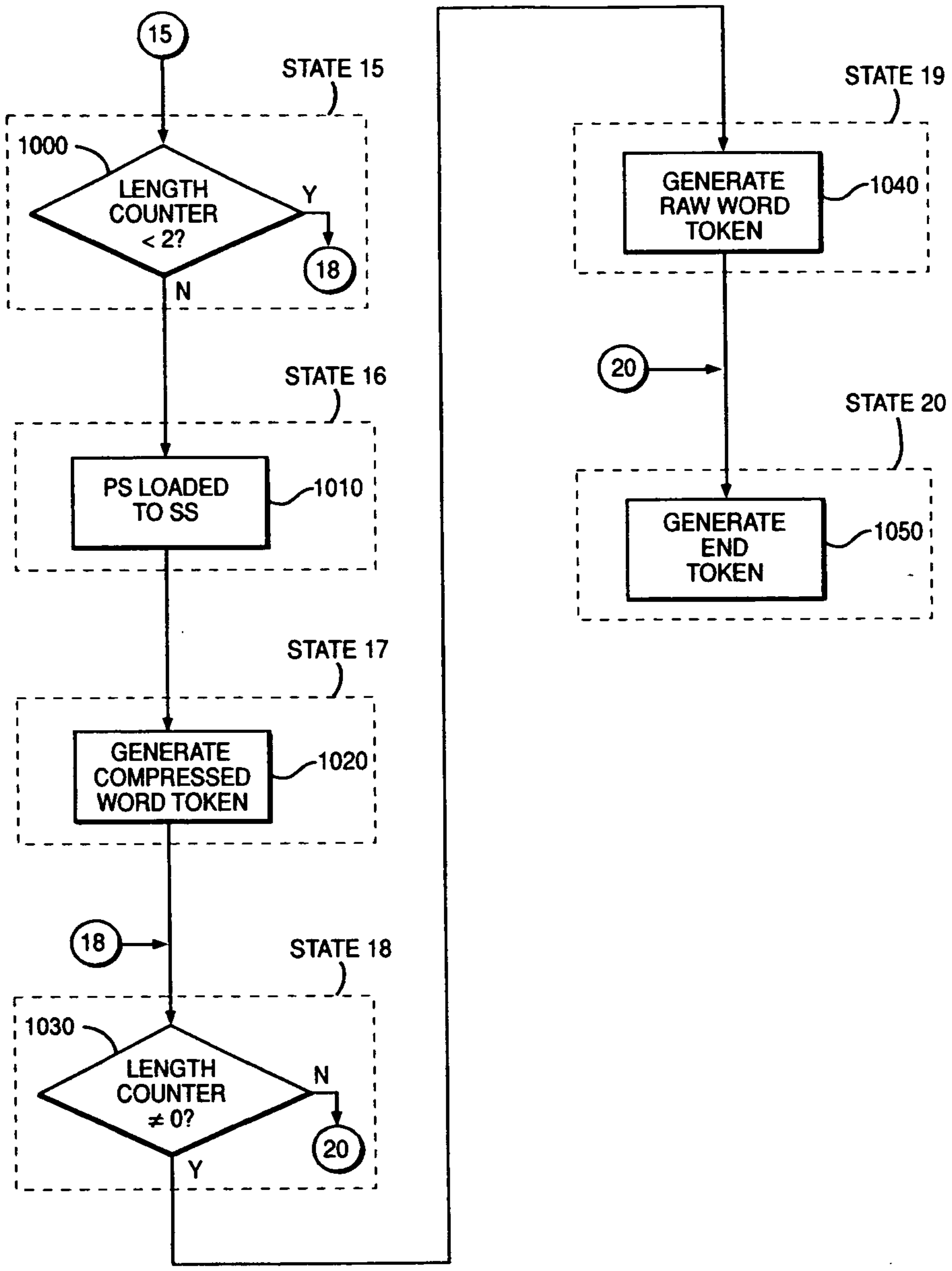

15
STATE 15
1000
LENGTH COUNTER < 2?
Y
18
N
STATE 16
PS LOADED TO SS
1010
STATE 17
GENERATE COMPRESSED WORD TOKEN
1020
18
STATE 18
1030
LENGTH COUNTER ≠ 0?
N
20
Y
STATE 19
GENERATE RAW WORD TOKEN
1040
20
STATE 20
GENERATE END TOKEN
1050

FIG. 6C

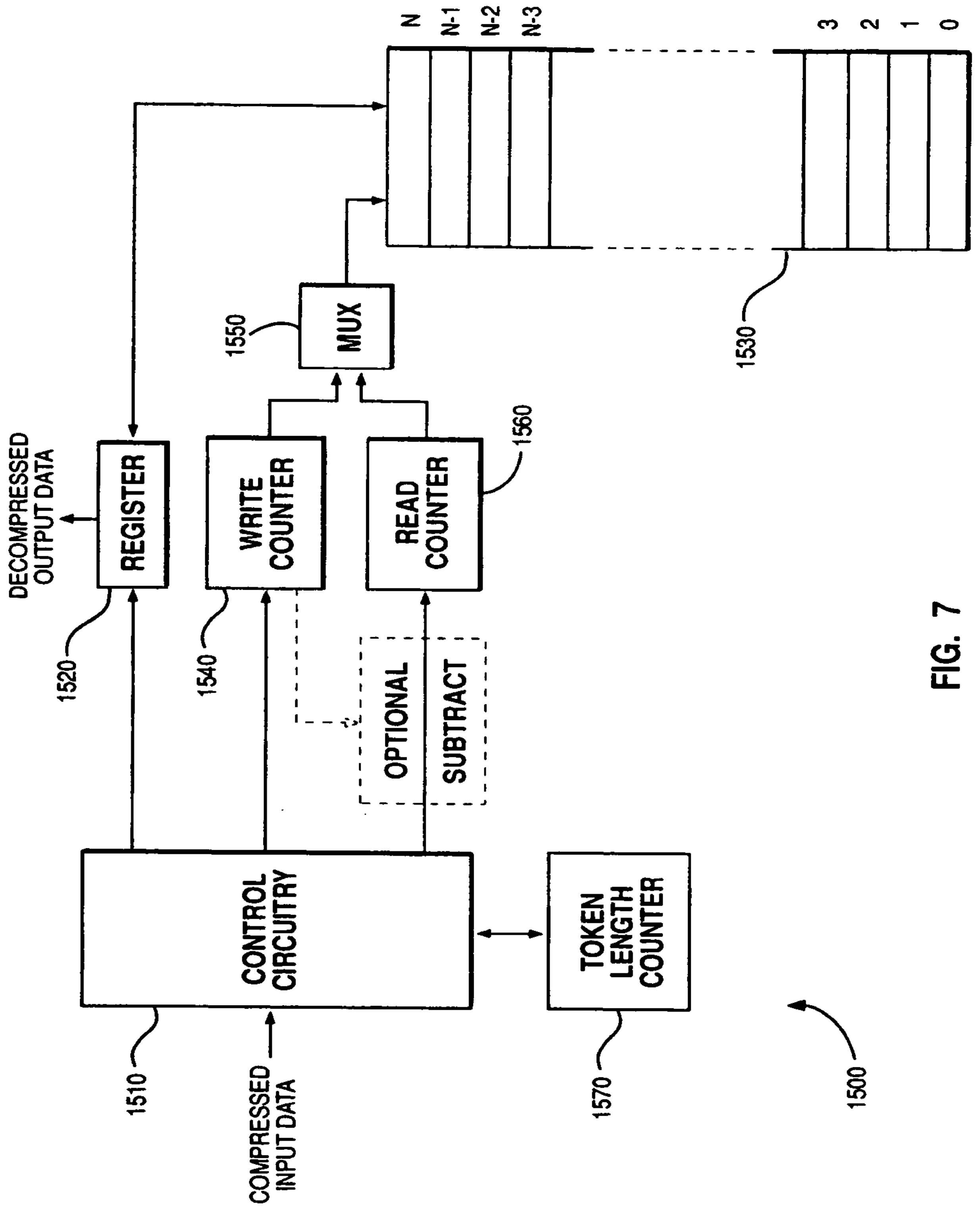

DECOMPRESSED OUTPUT DATA
REGISTER
WRITE COUNTER
READ COUNTER
MUX
N
N-1
N-2
N-3
3
2
1
0
OPTIONAL SUBTRACT
CONTROL CIRCUITRY
TOKEN LENGTH COUNTER
COMPRESSED INPUT DATA
1510
1520
1530
1540
1550
1560
1570
1500

FIG. 7